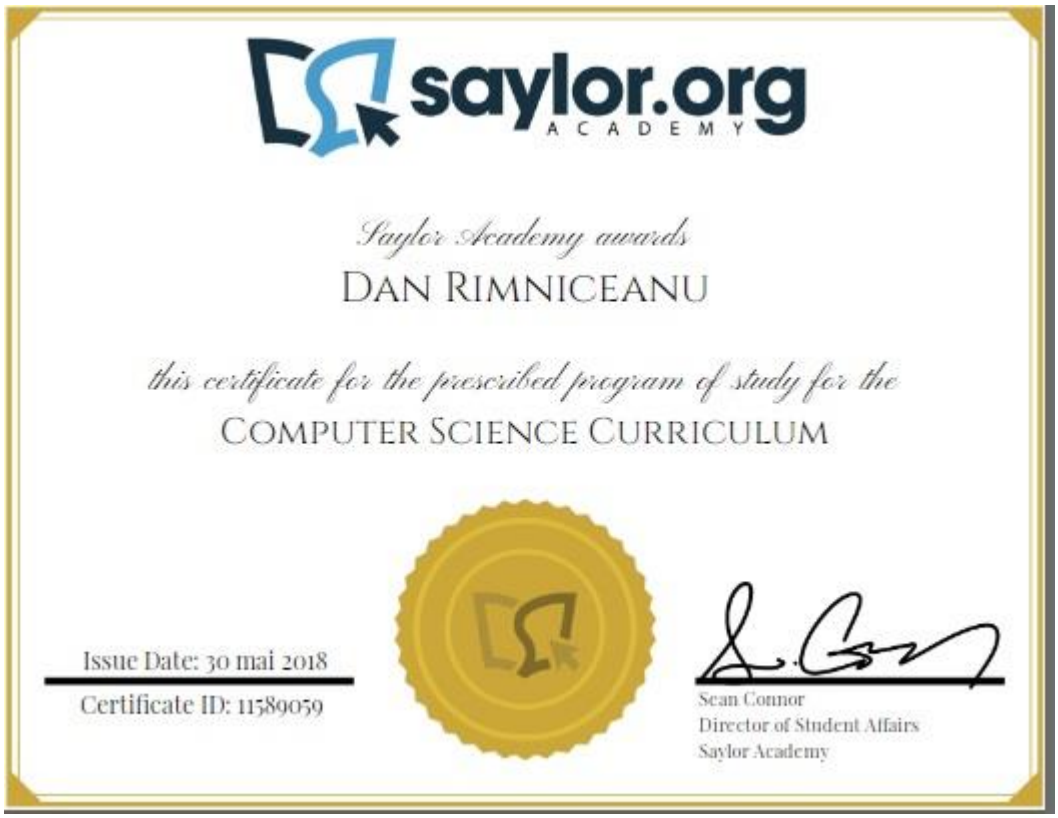


saylor.org
ACADEMY

Saylor Academy awards

DAN RIMNICEANU

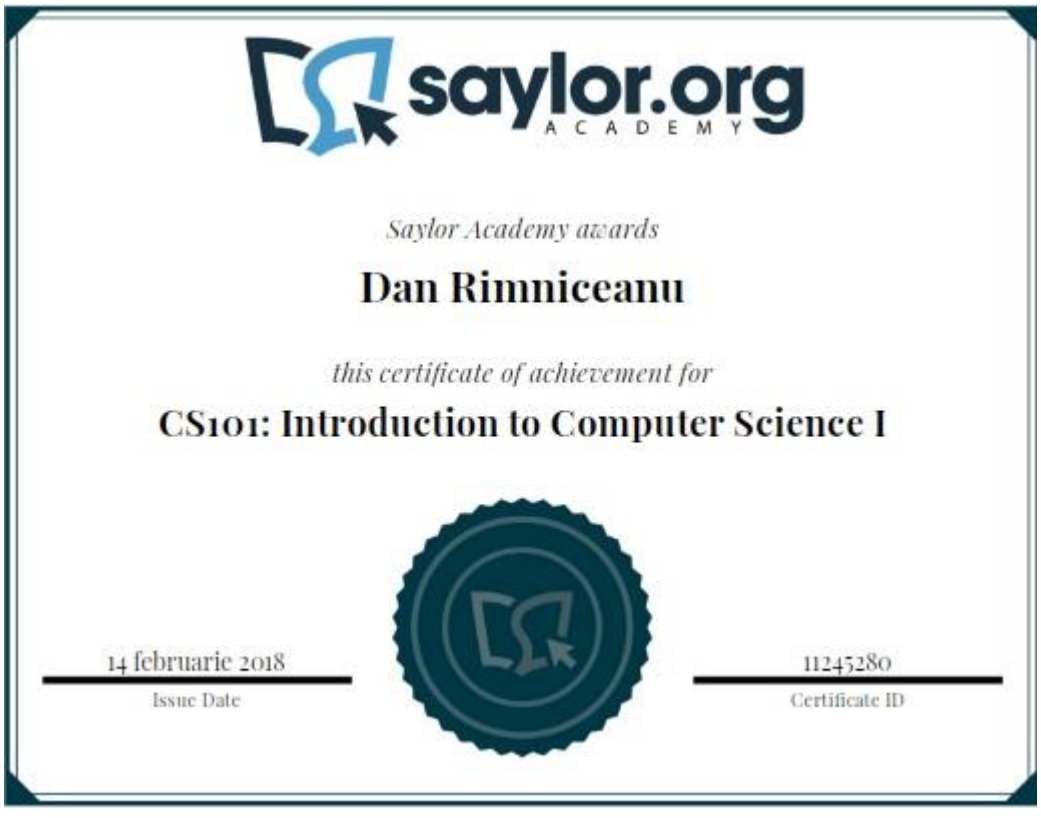
this certificate for the prescribed program of study for the

COMPUTER SCIENCE CURRICULUM

Issue Date: 30 mai 2018

Certificate ID: 11589059

Sean Connor
Director of Student Affairs
Saylor Academy



saylor.org
ACADEMY

Saylor Academy awards

Dan Rimniceanu

this certificate of achievement for

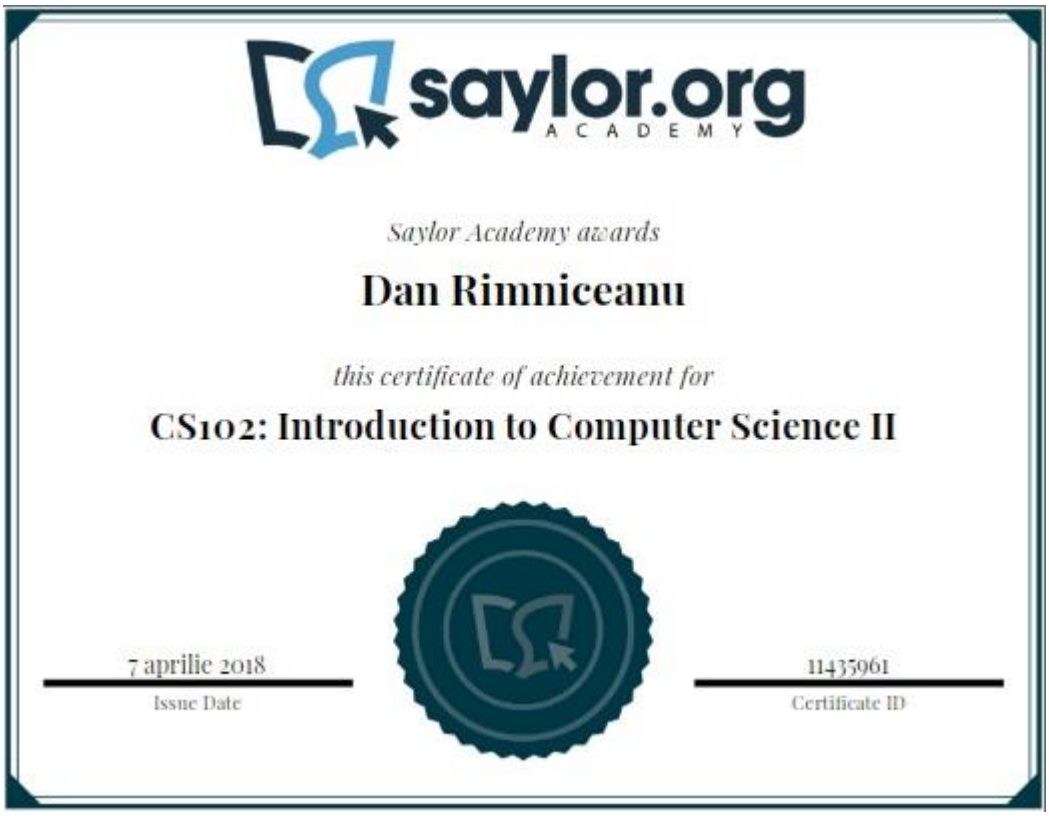
CS101: Introduction to Computer Science I

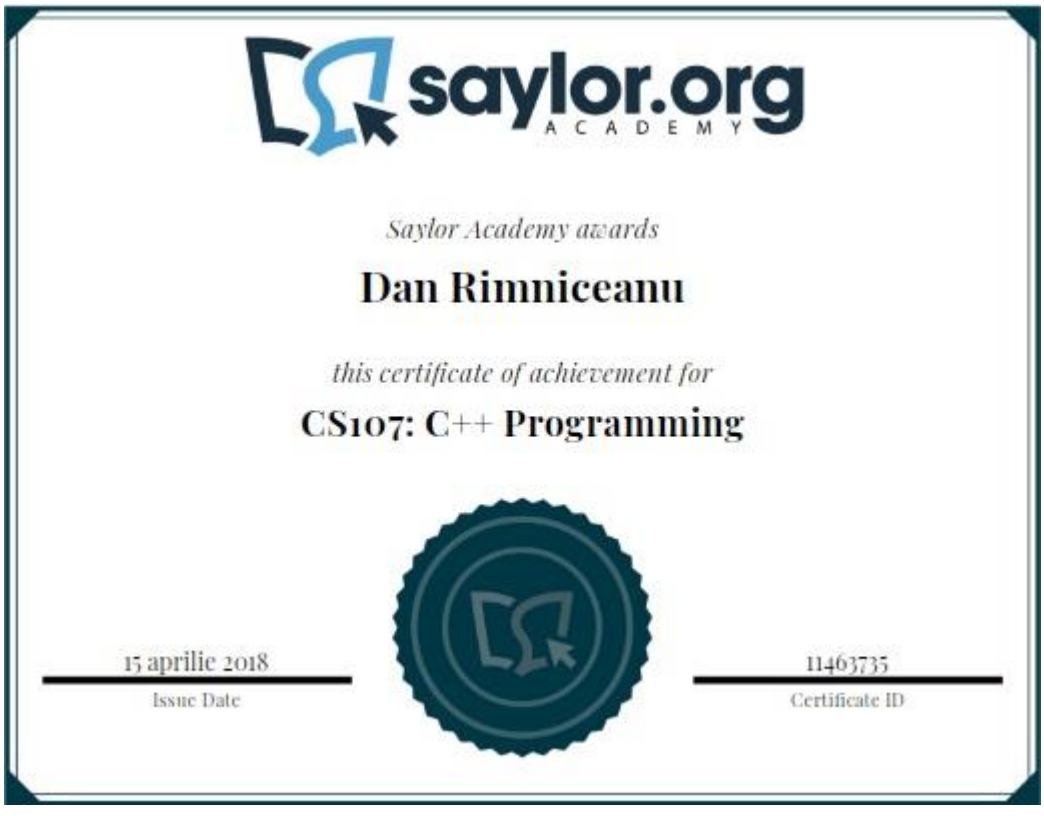
14 februarie 2018

Issue Date

11245280

Certificate ID


saylor.org
ACADEMY
Saylor Academy awards
Dan Rimniceanu
this certificate of achievement for
CS102: Introduction to Computer Science II
7 aprilie 2018
Issue Date
11435961
Certificate ID


saylor.org
ACADEMY
Saylor Academy awards
Dan Rimniceanu
this certificate of achievement for
CS107: C++ Programming
15 aprilie 2018
Issue Date
11463735
Certificate ID

This course will introduce you to the field of computer science and the fundamentals of computer programming. CS101 is specifically designed for students with no prior programming experience, and touches upon a variety of fundamental topics. This course uses Java to demonstrate those topics. Java is a high-level, portable, and well-constructed computer programming language developed by Sun Microsystems (now Oracle).
We begin this course with a brief history of software development, and show how human thought and computer programming are related. We build upon these general concepts to cover object-oriented programming terminology such as objects, classes, inheritance, and polymorphism. During this process, we use Java to show how those fundamentals are implemented in a real programming language. We do this by demonstrating Java's primitive data types, relational operators, control statements, exception handling, and file input/output.

By the end of the course, you will understand the basics of computer science and the Java programming language. The principles you learn here will be developed further as you progress through the computer science discipline.

Unit 1: Introduction

We begin this course by developing a motivation for learning programming concepts and by reviewing the history of computer programming languages, and show the connections between human thought and its expression in programming languages. We then discuss hardware – the physical devices that make up a computer – and software – operating systems and applications that run on the computer. We conclude with a brief discussion of the Java programming language, which we will use throughout the rest of the course. By the end of this unit, you will have a strong understanding of the history of programming, and you will be ready to learn about programming concepts in more detail.

Completing this unit should take you approximately 11 hours.

- Upon successful completion of this unit, you will be able to:
    - describe the history of software and computer design;
    - explain formal logic and how it is a subset of human logic;
    - describe the relationship between human logic, computers, and computer languages;
    - describe the basic computer model;
    - explain the programming life cycle;
    - discuss the history of the Java programming language;
    - set up a Java Development Kit (JDK) for Java development;
    - set up a NetBeans IDE for Java programming;
    - write and run a simple Java program; and
    - explain how computers are a tool to assist people and describe appropriate and inappropriate uses of computers.

- 1.1: History and Motivation
    - Computer HistoryBook

      These articles give some insight into the early history of computers, and introduce the powerful ideas that enabled the computer architecture of our day and that will influence computer architecture of tomorrow.

    - The History of Computing Hardware (1960-Present)Page

      Read this article to supplement the previous articles. As you see here, the history of computers is split into "generations". This course will primarily focus on the third generation of computers, which began in the 1960s, and the microcomputer technology of today, which has meant that computers are now present in the homes of many people across the world.

    - Tally Sticks and the AbacusPage

      Lest you get the impression that computers are a new phenomena, this short article demonstrates that computers, in one form or another, have been around since at least 18,000BC.

    - The History of Programming LanguagesPage

      This article focuses on the evolution of programming languages over the years. How did programming languages change as the "generations" of computers advanced?

- 1.2: Reflecting Human Thought via Computers
    - A Concise Introduction to LogicBook

      Aristotle lived in ancient Greece around 384-322 BC. He is credited with the invention of a field of philosophy called "formal logic". This is a way of thinking that only admits to something being absolutely true or absolutely false. For instance, an item is in one group or some other group but it can not be a member of more than one group at a time. Everything is black or white. There are no shades of gray. You may have heard this called "binary thinking". However, it is perfect for computers because computers are nothing more than black boxes filled with on/off switches that are either completely on or completely off. As such, there is nothing new about computers as we know them today. They simply allow us to mechanize complex constructs in formal logic, and to do so at very high speeds. Gottfried Leibniz (1700s) created a means of using binary values to perform arithmetic. Essentially, this is the translation of our usual Base 10 numeric values to Base 2 (binary) values, and the means to perform arithmetic operations on such values. George Boole (1815-1864) invented Boolean Algebra, a mathematical means of expressing and manipulating formal-logic variables using logical operators to get correct results in highly complex situations. So, briefly, you can see the philosophical and mathematical history that underlies modern computers. This history, and

the mechanization of that history by computers, allows human thought to be reflected and carried out consistently, although not perfectly. (Computers can not even perform the operation 1/3 with absolute accuracy.) For this introductory course, it is enough to give some thought to formal logic and its basic ideas. Read these two chapters.

- An Introduction to Formal LogicFile

  Supplement what you just read by reading the first chapter of this book. As you read, you will see the careful human thought that is required to create the logical constructs required to get the computer to do anything worthwhile. Section 1.6 begins the process of creating a formal syntax, or language, from human statements. In this course, we use the formal language Java to translate human statements into something that can be translated into the 1's and 0's the computer works with, the settings of the computer's electronic switches.

- 1.3: Why Computers and Computer Programming Matter
  - Innovation and Its ApplicationFile

    One definition of innovation is the process by which solutions are created for complex problems. Another perspective sees innovation as creating something entirely new that takes situations and people down a path not previously conceived. Read this section, which discusses innovation and several practical aspects of bringing new ideas to fruition, and demonstrates case studies of successful applications.
  - Using computers more widely has benefited society greatly. However, computers have their limitations, as do we humans. For all the hype about "intelligent" computers, it is humans who create computers, who put data into computers, who write programs that process that data, and who select and interpret the results of that processing. Clearly, the "intelligence" of computers comes directly from the humans who create and use them. Thus, we can never say, "Oh, It was just the computer. It made a mistake." Spend some time thinking about that idea before you move forward
- 1.4: Hardware/Software Model of a Computer
  - Introduction to Computer SystemsBook

    Review these notes. For maximum benefit, go through these notes interactively, thinking about and answering the questions at the bottom of each page. These notes are an experiment in applying the "programmed learning" method to web-based computer aided instruction. The subject is Java Programming for beginning programmers. The content is intended to start beginning programmers out on the track to professional-level programming and reinforce learning by providing abundant feedback. Java is a programming language that is used often in professional practice.

- The ProcessorBook

  Review these notes. As you work through them, think about and answer the questions at the bottom of each page.

- Introduction to Number Systems and BinaryPage

  Watch this lecture on binary numbers, which are used to represent numbers in computer memory. In fact, all types of information, not just numbers, but characters as well, are stored in memory using binary bits (that is, digits) of 0 and 1. We still write numbers in our programs using decimal numbers, but the interpreter converts them to binary when it interprets them for execution as part of a machine language instruction (the interpreter interprets the Java statement to equivalent machine language statements).

- Quiz on Computer Systems and the Processor

  Take this short quiz to see how well you understand what we've covered so far.

  - This quiz **does not count towards your grade**. It is just for practice!
  - You will see the correct answers when you submit your answers. Use this to help you study for the final exam!
  - You can take this quiz as many times as you want, whenever you want.

- 1.5: The Software Development Lifecycle

  - The Programming LifecyclePage

    Read this brief article about the programming lifecycle. As you read, consider the following steps of the programming lifecycle: Planning and Analysis, Design, Implementation, Testing and Debugging, Deployment, Maintenance

  - Comparing Waterfall, Unified, and Agile Software Development ProcessesPage

    Three major approaches to software development, which are the means of carrying out the programming lifecycle, have been tried by practitioners. This article compares those three. The most used today are Unified and Agile. These processes help to guide software development projects. They are not software development in and of themselves. Always keep in mind that the goal is to deploy excellent software, not to follow processes.

- 1.6: Overview of Java

  - Translators: Compilers and InterpretersPage

    This article explains the distinction between compliers and interpreters. It is important that you understand the difference.

  - A Top-Level View of JavaPage

Java is both a compiled and interpreted language. What you write as Java code is compiled into machine-independent bytecode. The bytecodes are interpreted for the particular machine the bytecodes are running on. In this way, Java becomes machine independent. Such is not the case for languages like C/C++, where the language code is compiled into assembly code for a particular machine. A linker turns that code into an executable for that machine.

- Downloading and Installing JDKBook

  This page shows how to download and install the latest version of the Java Development Kit (JDK). Read the instructions carefully to set the "classpath" mentioned in Step 3. Once JDK has been installed, you can write a simple Java program using an editor such as notepad and run it from a command prompt. Alternatively, Java programs can be written using an Integrated Development Environment (IDE) such as NetBeans, described below.

- Downloading and Installing NetBeans IDEBook

  These instructions describe how to download and install NetBeans, a commonly used IDE for Java programming. Using an IDE means that you have all of the tools you need in one place (your "development environment") instead of having to organize things manually. Use the instructions in Step 2 to write a simple Java program called "Hello.java" and then compile and run it.

- Introduction to JavaBook

  Work through these slides. As you read, think about and answer the questions at the bottom of each page. These will be your first experience with Java, so make sure you follow each step closely.

- Introduction to Java – PracticeURL

  Think of the word that should fill in the blank for each statement. Click on each one to see what the right answer should be.

Unit 2: Object-Oriented Programming

Java is an object-oriented programming language. Object-oriented (OO) programming has proven to be one of the most effective and flexible programming paradigms. This unit will begin with a discussion of what makes OO programming so unique, and why its advantages have made it the industry-standard paradigm for newly designed programs. We then discuss the fundamental concepts of OO and relate them back to Java. By the end of this unit, you will have a strong understanding of what OO programming is, how it relates to Java, and why we use it.

Completing this unit should take you approximately 3 hours.

- Upon successful completion of this unit, you will be able to:
  - discuss the differences between object-oriented and procedural languages;
  - explain the difference between classes and objects; and
  - explain object-oriented concepts such as inheritance, encapsulation, and polymorphism.

- 2.1: Fundamental Concepts of OO Programming
  - Traditional vs. Object-Oriented ApproachesBook

    Object-oriented approaches to software development are an important expansion of procedural approaches. Java explicitly supports both approaches, but you should focus on the object-oriented approach. This article compares the two approaches and explains the fundamentals of each.

  - Advantages and Disadvantages of Object-Oriented ProgrammingPage

    Read this article about object-oriented programming. Every paradigm has its advantages and disadvantages. OO is the same.

- 2.2: Using Java for OO Programming
  - Objects and Object-Oriented ProgrammingPage

    This section explains fundamentals of object oriented programming. As you read, focus on the difference between classes and objects.

  - Inheritance, Polymorphism, and Abstract ClassesPage

    This section expands the discussion from objects and classes to inheritance, polymorphism, and abstract classes. These facets of OO programming are a natural consequence of the basic concepts behind objects and classes.

  - Java EncapsulationPage

    Encapsulation facilitates control over access to components of a class. Read this article to get a clear understanding.

  - DecouplingPage

    A major principle of OO programming is keeping the implementation of an object separate from manipulation. This article explains how that works.

Unit 3: Java in Practice

Now that you have a basic understanding of object-orientation, we'll move on to the practicalities of Java, which is the programming language we'll be studying. The Java-related concepts you will learn in this unit are in many cases directly transferable to a

number of other languages. We will begin by learning about "Hello World", a basic software application that simply prints "Hello World" to the screen as a means of demonstrating the most essential elements of a programming language, and will then move on to discuss variables, literals, data types, and operators. In addition, we will also learn about two different styles of adding comments to the code. By the end of this unit, you should have an understanding of Java basics and be prepared to apply those concepts later in the course.

Completing this unit should take you approximately 9 hours.

- Upon successful completion of this unit, you will be able to:
  - write Java classes using file naming conventions;
  - use pre-written Java classes from various packages in the Java API;
  - explain and use primitive data-types in Java;
  - declare and use variables of different data types while writing Java programs;
  - perform various mathematical operations using +, -. *, /, and % operators; and
  - use String class and its methods while writing Java programs.

- 3.1: Compiling and Executing a Java Program
  - Running Example ProgramsBook

    This chapter provides step-by-step instructions of writing a Java program using a text editor, and then compiling and running this program from command prompt. Several Integrated Development Environments (IDEs) are mentioned in this chapter. We have already installed NetBeans. That and Eclipse are the most popular in Java professional practice.

  - Review – Compiling and Executing a ProgramURL

    Use this page to quickly review what you've learned so far.

- 3.2: Working with Classes
  - Small Java ProgramsBook

    This chapter discusses naming and coding conventions as well as reserved words in Java. When you go through this chapter, you'll get some hands-on experience with writing in Java.

  - Review – Small Java ProgramsURL

    Use this page to quickly review what you've learned so far.

- 3.3: Importing Libraries

- Importing Libraries in JavaPage

  Java classes and interfaces can be organized into packages to group related types and for name-space management.

- 3.4: Primitive Data Types
  - Primitive DataBook

    This chapter discusses eight primitive data types in Java.

  - Flash Cards on Primitive Data TypesURL

    Use this page to quickly review what you've learned so far.

- 3.5: Basic Operations in Java
  - Assignment OperatorsPage

    Read this introductory article.

  - Variables and Assignment StatementsBook

    Read this chapter, which covers variables and arithmetic operations and order precedence in Java.

  - Expressions and Arithmetic OperatorsBook

    Read this chapter, which discusses arithmetic operations in greater detail along with solving expressions with mixed data types.

- 3.6: The String Class
  - Strings in PythonPage

    Read this section, which introduces the concepts of strings using Python.

  - Strings and Object References in JavaBook

    The String class is used for text manipulation. As you read, you will learn different ways to create Strings, methods to manipulate Strings, the String concatenation operator '+', and about how Strings are immutable.

  - More about StringsBook

    This chapter accompanies the previous one. Read it for even more info on Strings.

Unit 4: Relational and Logical Operators in Java

In this unit, we discuss relational and logical operators in Java, which provide the foundation for topics like control structures that we will discuss later in the course. In this unit, we start by taking a look at operator notation. We then discuss relational operators as they apply to both numeric operands and object operands. The unit concludes with an introduction to logical operators. By the end of this unit, you should be able to perform comparisons and logic functions in Java and have a fundamental understanding of how they are employed.

Completing this unit should take you approximately 5 hours.

- Upon successful completion of this unit, you will be able to:
  - explain relational operators such as >,>=, <, <=, ==, and !=;
  - write Boolean expressions using relational operators;
  - use logical operators such as &&, ||, and !; and
  - evaluate truth-tables.

- 4.1: Relational and Logical Operators

  - Boolean ExpressionsBook

    This chapter introduces relational and logical operators. It also discusses the use of these operators to write Boolean expressions.

  -  Java Data and OperatorsFile

    This chapter goes into more depth about relational and logical operators. You will have to use these concepts to write complex programs that other people can read and follow.

  - More about Objects and ClassesBook

    The relational operations on primitive data are ==, >=, <=, >, <, and !=. They compare two data values, when those values' type has an ordering. For example, integers are ordered by size or magnitude. The result of a relational operation is a boolean value: either True or False. The relational operators on objects like Strings are different, and they are expressed as methods. Pay special attention to the equality method, equals().

  - Comparable InterfaceBook

    Objects that have an ordering are compared using the compareTo() method.

  - Review - Boolean ExpressionsURL

    Complete this review exercise. Think of the response to fill in the blank or to answer true/false, or write your response down. Then, click on the box to reveal the answer to each question.

- 4.2: Truth Tables
    - Truth Tables and De Morgan's RulesBook

      Read this chapter, which discusses Boolean variables as used in decision tables. Truth tables are used to collect variables and their values relative to decisions that have to be made within control structures.
    - Review - Truth Tables and De Morgan's RulesURL

      Answer the questions in this review. After you have thought of your answer, only then click on the box in each question to reveal the correct answer.

Unit 5: Control Structures

Control structures dictate how a program will behave under certain circumstances. Control structures belong to one of two families: those that test values and determine what code will be executed based on those values, and those that loop, performing identical operations multiple times. Control structures like if-then-else and switch the program to behave differently based on the data that they are fed. The while and for loops allow you to repeat a block of code as often as it is needed. As you will see, that functionality can be very useful when designing complex programs.

This unit will introduce you to control structures and how they are used before moving on to discuss if, switch, while/do-while, and for loops. We will also discuss some advanced topics, such as nesting and scope. By the end of this unit, you should be able to draw from the information you learned in the previous unit to create a control structure, which will allow you to create more involved and useful programs.

Completing this unit should take you approximately 10 hours.

- Upon successful completion of this unit, you will be able to:
    - use control structures;
    - write an 'if' statement;
    - write a 'switch' statement; and
    - explain various looping structures such as for, while, and do loops.

- 5.1: Introduction to Control Structures
    - Control StructuresPage

      This article gives a general introduction to control structures.

- 5.2: The 'if' statement
    - Decision MakingBook

Read this chapter, which reviews how computers make decisions using if statements. As you read this tutorial, you will understand that sometimes it is important to evaluate the value of an expression and perform a task if the value comes out to be true and another task if it is false. In particular, try the simulated program under the heading "Simulated Program" to see how a different response is presented to the user based on if a number is positive or negative.

Pay special attention to the "More Than One Statement per Branch" header to learn how the 'else' statement is used when there is more than one choice.

- Quiz on the If Statement

Attempt this ungraded quiz.

- 5.3: The 'switch' Statement
  - The Conditional Operator and the 'switch' StatementBook

Read this chapter, which discusses the switch and '?' operators to write conditional statements. As you read this tutorial, you will learn that sometimes it is better to use a 'switch' statement when there are multiple choices to choose from. Under such conditions, an if/else structure can become very long, obscure, and difficult to comprehend. Once you have read the tutorial, you will understand the similarity of the logic used for the 'if/else' statement, '?', and 'switch' statements and how they can be used alternately.

  - Quiz on the Conditional Operator and the Switch Statement

Attempt this ungraded quiz.

- 5.4: The 'while' and 'do-while' Loops
  - Loops and the While StatementBook

Read this chapter, which explains while loops. This is in contrast to how a do-while loop works, which we will discuss later. In a do-while loop, the body of the loop is executed at least one time, whereas in a while loop, the loop may never execute if the loop condition is false.

  - Review - Loops and the While StatementURL

Complete the review exercise. Think of the correct response to fill in the blank for each question, and then click on the blank to reveal the correct answer.

  - The Do StatementBook

The 'do-while' loop is a variation of the while loop. 'do-while' loops always execute at least once, whereas while loops may never execute.

- Quiz on the Do Statement

  Attempt this ungraded quiz.

- 5.5: The 'for' Loop
  - The For StatementBook

    The 'for' loop is more compact than the 'while' and 'do' loops and automatically updates the loop counter at the end of each iteration. Both 'for' and 'while' loops are designed for different situations. You'll learn more about when to use each later.

  - More about the For StatementBook

    This chapter discusses the 'for' loop in greater detail, as well as the scope of variables in the 'for' loop.

  - Quiz on For Loops

    Attempt this ungraded quiz.

  - Quiz on Further For Loops

    Attempt this ungraded quiz.

- 5.6: Advanced Topics
  - Nesting Loops and IfsBook

    This chapter discusses how control structures such as loops and if statements can be combined together to implement program logic.

  - Blocks, Loops, and BranchesPage

    This article explains variable scope within different control structures such as a block, while loop, and branching statement such as the if statement.

  - Review - Nested Loops and IfsURL

    Complete this review exercise. Think of the correct response to fill in the blank, and then click on the blank to reveal the correct answer.

Unit 6: User-Defined Methods

In addition to the methods predefined in Java, we can write user-defined methods. In this unit, we will discuss how to name a method, declare a parameter list, and specify the return type. This unit introduces the scope of variables as well. By the end of this unit, you will have a strong understanding of how to define and call a method.

Completing this unit should take you approximately 4 hours.

- Upon successful completion of this unit, you will be able to:
  - write methods with zero or more parameters; and
  - make method calls.

- 6.1: Creating and Using Methods

  - Methods: Communicating with ObjectsBook

    We communicate with objects using methods. Methods are executable code within each object, for which an interface has been established. Sometimes the interface is only for the object itself. Other times it is an interface accessible by other objects. This chapter discusses that topic in detail.

  - Threads and Concurrent ProgrammingBook

    Threads may be seen as methods that execute at "the same time" as other methods. Normally, we think sequentially when writing a computer program. From this perspective, only one thing executes at a time. However, with today's multi-core processors, it is possible to literally have several things going on at the very same time while sharing the same memory. There are lots of ways that this is done in the real world, and this chapter goes over them in a way that you can apply to your own projects.

- 6.2: Overloaded Methods

  - Parameters, Local Variables, and OverloadingBook

    This chapter reviews method parameters and local variables, as well as method overloading and method signature.

    Method overloading means two or more methods have the same name but have different parameter lists: either a different number of parameters or different types of parameters. When a method is called, the corresponding method is invoked by matching the arguments in the call to the parameter lists of the methods. The name together with the number and types of a method's parameter list is called the signature of a method. The return type itself is not part of the signature of a method.

  - Review - Parameters, Local Variables, and OverloadingURL

    Complete this review exercise. Think of the correct response to fill in the blank, and then click on the blank to reveal the correct answer.

Unit 7: Arrays

An array is a multi-dimensional fixed-size data structure that allows elements of the same data type to be stored in it. Each array element has a unique index associated with the value it stores. This unit introduces two-dimensional arrays and their applications.

Completing this unit should take you approximately 6 hours.

- Upon successful completion of this unit, you will be able to:
  - declare and use one-dimensional arrays;
  - create, initialize, and access multi-dimensional arrays; and
  - use enhanced for-loops to iterate over an arrays' elements.

- 7.1: Introduction to Arrays
  - ArraysBook

    This chapter introduces arrays, a common multi-dimensional data structure used to store data of the same type. (Note that a class is a "data type" in this sense.) Arrays have some number of dimensions and a number of elements in each dimension. These are established when the array is created. The range of the array's index in each dimension goes from zero to the number of that dimension's elements minus one. A for loop is commonly used to initialize, manipulate, and access the values in an array. Here, we treat one-dimensional arrays, sometimes called vectors.

  - One-Dimensional ArraysPage

    This tutorial gives more examples of how to create, initialize, and access one-dimensional arrays. It also briefly illustrates the use of arrays to reference class objects.

- 7.2: Two-Dimensional Arrays
  - Two Dimensional ArraysBook

    This chapter expands our discussion on one-dimensional arrays to two-dimensional arrays. A two-dimensional array is a data structure that contains a collection of cells laid out in a two-dimensional grid, similar to a table with rows and columns although the values are still stored linearly in memory. Each cell in a two-dimensional array can be accessed through two indexes that specify the row number and column number respectively. Like s one dimensional array, the range of each index is from zero to the size of the row or column minus one. A nested for loop is commonly used to initialize, manipulate, and access the values in a two-dimensional array.

  - Quiz on 2D Arrays

    Attempt this ungraded quiz.

- Multi-Dimensional ArraysBook

  We can expand the whole idea of arrays to multiple dimensions, beyond one or two dimensions. Read these sections to see how this is done.

- 7.3: Common Array Algorithms
  - Common Array AlgorithmsBook

    This chapter discusses how a for loop can be used to iterate over the elements of a one-dimensional array. In addition, it also discusses enhanced for loops. The chapter demonstrates the use of arrays to solve common problems such as finding sum, average, maximum, and minimum values of numbers stored in array. Pay attention to some of the common programming errors one can make while using arrays.

  - ArrayLists and IteratorsBook

    Read this chapter about lists and the useful methods that are available for adding and inserting elements into a list.

  - Java Program to Multiply Two MatricesPage

    Study this program to get a sense of how to manipulate two-dimensional arrays.

Unit 8: Java I/O and Exception Handling

In this unit, we discuss two important programming concepts in Java: input and output. Input and output techniques allow programmers to connect the virtual world of computers to the real world. Because of this, you must fully understand how to use a programming language's built-in I/O (input/output) functionality. In this unit, we discuss function I/O before moving on to file I/O (writing to and reading data from files). Each unit contains a discussion of the applicable Java classes, which are part of the standard programming language (FileWriter, PrintWriter, FileReader, BufferedReader, IOException). We then identify the common pitfalls and design concepts that you should keep in mind as a programmer. By the end of this unit, you will have a strong understanding of how to write and read from a file and how to write a Java program that performs these functions.

Completing this unit should take you approximately 6 hours.

- Upon successful completion of this unit, you will be able to:
  - describe the Java I/O package;
  - read and write data from/to an external file;
  - use the Java I/O package to retrieve data for populating method parameters;
  - explain error-handling via exceptions; and
  - apply exception handling techniques.

- 8.1: Input/Output in Java
    - Input and OutputBook

      Java provides a Scanner class to facilitate data input/output. In this section, you will learn about the Scanner class that is used to get input from the user. Java also defines various methods from the Scanner class can convert user input into appropriate data types before conducting any operation on the data.

    - Review - Input and OutputURL

      Complete this review exercise. Think of the best response to fill in the blanks for each question, and then click on the blank in each question to reveal the correct response.

    - String FormationPage

      Read this article to learn how to format your output for specific purposes.

    - How to Write Data to Console in JavaPage

      Read this tutorial.

- 8.2: Writing Data to a File
    - Input and Output StreamsBook

      This chapter explains how input and output streams can be used for writing files, similar to how they are used for writing to the display.

    - Writing Text FilesBook

      This chapter explains Java's FileWriter class and how you can use it to store data in files.

    - Quiz on Writing Text Files

      Attempt this ungraded quiz.

- 8.3: Reading Data from a File
    - Reading Data from a FileBook

      This chapter discusses Java's FileReader and BufferedReader classes in detail. FileReader and BufferedReader classes are used together when reading data from an external file. The use of the BufferedReader class allows data to be buffered as it is read from a file before manipulating it. The readLine() method of the BufferedReader class reads a line of text from a character-oriented input stream, and puts it into a new String object.

- Quiz on FileReader and BufferReader

  Attempt this ungraded quiz.

- 8.4: File Input for Method Arguments
  - java.io.File and File InputBook

    Section 4.6 describes and illustrates how to get the values of method arguments from a file.

- 8.5: Handling Exceptions
  - Exceptions: When Things Go WrongBook

    It is not a matter of IF but WHEN things will go wrong in a computer program. Sometimes there are bugs, errors of one form or another. There also also unforeseen use cases. You can never assume a computer program is perfect. Exception-Handling helps us to catch erroneous events and devise means of correcting them. We discuss this topic here since exception-handling can take more code than should be put into the main line of execution. In such cases, a method in an exception-handling class should be called. Exception handling mechanisms allow a program to continue executing, instead of terminating it abruptly, even if an error occurs in the program.

This course builds on what we learned in CS101: Introduction to Computer Science I. It will introduce you to a number of more advanced Computer Science topics, laying a strong foundation for future academic study in the discipline. We will begin with a comparison between Java, the programming language used in the previous course, and C++, another popular, industry-standard programming language. We will then discuss the fundamental building blocks of Object-Oriented Programming, reviewing what we have already learned and familiarizing ourselves with some more advanced programming concepts. The remaining course units will be devoted to various advanced topics, including the Standard Template Library, Exceptions, Recursion, Searching and Sorting, and Template Classes. By the end of the course, you will have a solid understanding of Java and C++ programming, as well as a familiarity with the major issues that programmers routinely address in a professional setting.

Unit 1: C++ and Java

Having completed CS101: Introduction to Computer Science I, you should have a strong grasp of Java and its uses and a basic understanding of Object-Oriented Programming. This course will employ both Java and C++, another industry-standard programming language. In this unit, we will outline the similarities and differences between these two languages, noting how each is used in the industry. We will also take an in-depth look at the history, importance, and functionality of C++ and compare the basic building blocks of each language in order to draw a distinction between the two and further acquaint you with both languages.

Completing this unit should take you approximately 4 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the history of C++ and the language's development;
  - demonstrate an understanding of how C++ and Java are used in industry; and
  - demonstrate an understanding of the similarities and differences between Java and C++.

- 1.1: History of C++
  - 1.1.1: Development of C
    - William Stewart's "C Programming Language History"Page

      Read this history of C to see where the C fits into the 'larger picture' of computing history. The history of C is part of the lore of programming, Unix, and the Internet, which everyone who studies programming should know.

- 1.1.2: Branching C to C++

  - Wikipedia: "Compatibility of C and C++"Page

    This article discusses the compatibility of C and C++. Compatibility of two programming languages refers to the extent to which a program written in one of the languages can be used without modification in the other. Compatibility includes both syntax (grammar) and semantics (the execution of grammatical statements). C and C++ have a degree of upward-compatibility, but there are differences since they are distinct languages that have evolved separately.

- 1.1.3: History of Object-Oriented Programming (OOP)

  - *Ruby for Beginners*: "Object-Oriented Programming"Page

    To understand a language, it helps to know what motivated its development, its principle concepts (called a "programming paradigm"), and how it relates to other languages. This page explains the principle concepts and paradigm of Ruby, an object-oriented ("OO") language developed in the mid-1990s. The concepts explained here also apply to other OO languages.

    You can learn OO programming via a course on an OO language, which will emphasize the syntax and features of that language. OO features that are not implemented in that language or different implementations of a given feature may not be covered. They would typically be encountered when you learn a different OO language. In this course, we teach the foundational concepts of the OO paradigm, and use various languages, particularly C++ and Java, to demonstrate them.

- 1.2: C++ and Java in Industry

- 1.2.1: Influence of Prevalence of C

  - Jeremy Hansen's "The Rook's Guide to C++"Page

    Read Chapter 1, which discusses the development relationship of C and C++. If you wish, take a look at the table of contents of the full text, which can be found here. Most of the content comes from C, as much of C is included in C++, though C++ adds additional features such as classes and objects.

- 1.2.2: Java Overview

  - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

    Read Chapter 1, which describes how a 'system' can solve many types of problems. A 'system' consists of a computer (hardware components that carry out machine language instructions), software (programs written in a programming language, in particular Java), a communications interface (that interconnects the computer to a worldwide network of other computers), and an interface (that

enables users to access data from and run programs on many of the computers in the network).

While the operation of 'the system' applies to many programming languages, this chapter points out features of Java that improve the operation of the 'system', such as device independence via the Java Virtual Machine, OO, reusable class libraries (for user interfacing, event handling), network support, support for other technologies, and suitability for programming other devices.

- 1.2.3: C++ Overview
  -  Massachusetts Institute of Technology: John Guttag's "Introduction to C++"URL

    This article provides an overview of the elements of C++; specifically, the 'C' portion of C++.

    Note how section 2.2 describes tokens as the "minimal chunks of a program". The root goal of programming is solving problems using the 'chunks' of a programming language. Of course, the chunks must be appropriate for the type of problems to be solved. Generally, smaller chunks are applicable to many types of tasks, but involve more effort; larger chunks involve less effort, but are designed for more specific tasks.

    The following resource addresses 'larger' chunks available in C++.

  - Wikiversity: "C++"Page

    Solving problems with programs is made easier if we can reuse the same or similar solutions that already exist. We do this using the 'chunks' provided by a language, as described in the previous resources. These sections describe the larger 'chunk' features of C++. Larger 'chunks' consist of programming statements used to write a program and to complete programs or portions of programs that reside in libraries.

    The section "Classes and Inheritance" explains and illustrates classes, which enable reuse of large sections of programming code. "Templates" explains and illustrates generic programming using templates. Focus specifically on the Introduction, Function Templates, and Class Templates. It also discusses STL, the standard C++ library. Note that 'list' is a template in C++.

- 1.3: C++/Java Comparison
  - Wikipedia: "Comparison of Java and C++"Page

    When comparing two programming languages, consider their underlying concepts (goals, principles, model, paradigm), their syntax (grammar), their semantics (what tasks the language can instruct a computer to do), and what support (resources, libraries, tools, etc.) they provide. These considerations can be broken down into a list of specific features that are used to evaluate and compare

the two languages. The table in this article describes the similarities and differences between Java and C++.

- Hal Smith's "C++ vs. Java: Code to Executable"Page

  This video illustrates the different operational processes (compiling and linking) used in C++ and Java. Most of the video discusses the processes for C++, because it is more complicated than that of Java. The Java processes were described more thoroughly earlier in the course.

Unit 2: The Building Blocks of Object-Oriented Programming

Now that you are familiar with both C++ and Java, you are ready to explore more advanced topics in Object-Oriented (OO) Programming. We will begin by discussing the motivation behind programming with objects, learning the essential characteristics of OO Programming languages and identifying the advantages and disadvantages of various major programming frameworks. The unit will also provide a general history of OO Programming and, finally, review major characteristics of OO Programming. By the end of this unit, you will be able to discuss different programming paradigms and identify the main properties of OO Programming.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the various programming paradigms within computer science; and
  - demonstrate a detailed understanding of the principle concepts involved in object-oriented programming without regard to any particular programming language.

2.1: Programming Paradigms

- Marco Bonzanini's "Functional Programming in Python"Page

  The general computing paradigm has several states: the task to be performed; a strategy for how the task will be performed; transformation of the strategy into a detailed strategy that corresponds to computations that a computer can perform; implementation of the strategy as a set of instructions that can be executed by a computer; and, lastly, validation that the results of the execution perform the task in a satisfactory manner. In software engineering, these states are called requirements analysis, architecture and design, program code, and validation. This generic model has been refined into programming paradigms to support the transition from design to programming.

  This unit introduces several programming paradigms and explains main concepts of the Object-Oriented programming paradigm. Each paradigm has several, in some cases, many, programming languages that support it. This page illustrates

functional programming via Python. Note that it mentions some key features, including state, immutability, first class data type, recursion, anonymous functions, and lazy-evaluation.

- Allen Yip's "Prolog, Logic Programming and Programming Paradigm"Page

This page explains logic, functional, imperative, and object-oriented paradigms. It indicates that a programming language, while supporting a particular paradigm, typically may support other paradigms as well. With respect to the general computational paradigm, programming paradigms correspond to design and programming. This page mentions problem solving paradigms that support requirement analysis: lambda calculus, first order logic, and Turing machines. These are mathematical models developed to solve particular types of problems. Programming paradigms correspond to particular problem-solving models to support a particular type of task or to support many types of tasks. Programming paradigms can be thought of as models for transforming problem-solving models into executable programs.

- Massachusetts Institute of Technology: Dennis Freeman's "Object-Oriented Programming"Page

This video introduces the OO paradigm. The video mentions 4 modules: software engineering, signals, circuits, and planning. These modules focus on key concepts for building models for analyzing and solving problems that are applicable to many problems.

Our focus is on the software engineering module, which uses Python to illustrate the application of several concepts (modularity, abstraction, composition, and hierarchy) to programming as part of the OO paradigm. The video gives examples of data abstraction: using numbers and operations to form numeric expressions; strings and string operations to build string structures; and 'execution' abstraction (procedure names to represent sequences of instructions). Abstraction enables the composition of data to form more complex expressions and the composition of procedures to form hierarchies of procedures.

The video then shows how these two, data abstraction and procedure abstraction, are combined to form a composite structure, called a class, that consists of both data and procedures. An object is simply an instance of a class. A class contains the data and procedures common to the its instances, i.e. objects. An object consists of the class data, class procedures (called methods), assignment of values to the class (comm) data, and, can also include additional data and additional procedures. Just like data and procedures, classes and objects are given names, i.e. names are bound to them.

The video concludes with an explanation of how Python uses name bindings (the association of a name with a 'value' – a data value, expression, or procedure, object, or class instructions) when it executes instructions. Name bindings are stored by Python in a table called an environment. Each procedure, each class, and

each object has it own environment table. When a name is used in a procedure, Python looks up the name in the procedure's environment table to find the value associated with it. Since a Python name can be shared, for example, by a class and an object, rather than store the shared name twice the shared name is in the class environment and the object environment points to the class environment. Thus, a hierarchy of environments is used to represent a class-object hierarchy. The key concepts of modularity, abstraction, composition, and hierarchy are VERY important.

- 2.2: Fundamental Concepts of Object-Oriented Programming
    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

      Read Chapter 5: Objects and Classes.

      OO concepts of modularity, abstraction, composition, and hierarchy are very powerful and require intense attention to detail. The previous subunit on how Python implements name bindings gave a glimpse of the detail that is involved. This chapter gives a detailed presentation of OO in Java. The terminology in this reading is a little different; name-binding is called name scope. The chapter begins with an explanation of the data and procedures of a class (called class variables and class methods). The class data can be fixed for all objects, in which case, it is called static. Static variables are common to all objects. There is only one copy and, thus only one value, stored in the class.

      Recall from the Python OO overview that an environment associates a name with a 'value'. A static variable is in the class environment table, which points to the one copy. In contrast, a class can also have non-static variables and each object of the class contains its own copy of them. In the terminology for Java, each object has a name, which is a pointer to the location of the object's instance variables.

      The next detail to note is how objects are created and initialized (i.e. values assigned to its instance variables and its methods names) by assigning values to them in the class and by constructors. A consequence of the concepts of modularity and abstraction is reuse – in writing a OO program we can use classes that have been written by others and are part of the language or contained in class libraries associated with the language. Recall the generic computing paradigm consisted of several states: requirements, design, implementation, and validation. In software engineering, it is referred to as the program process.

      Section 5.3 gives insight into writing programs using classes. What classes, what objects, and how are they related? Are questions of program design. Section 5.4 illustrates the design and implementation stages of the process. The latter sections continue with the VERY important OO details of inheritance and polymorphism. They create a class hierarchy that enables code to be shared among classes and among similar, but different, objects. Whereas, Java has simple inheritance (a subclass can extend one superclass), C++ has multiple inheritance (

a subclass can extend 2 or more superclasses). Java does, however, have a restricted kind of multiple inheritance, called interfaces, where an interface can be implemented by 2 or more classes. As you finish the reading, you should appreciate how the concepts are connected. If you understand the variable names, in particular, the object names, 'this' and 'super', the class name 'Object', and the dot naming convention (e.g. ClassName.objectName.methodname), you should have a good understanding of the concepts and details presented in this reading.

- 2.2.1: Classes and Objects
  - Practice: Understanding Classes and Objects in JavaQuiz

    Complete this practice assignment. In this assignment there are two classes. One of the classes only consists of the main method (which is the procedure where the execution of the program begins). In this main method, an object of the other class is constructed.

- 2.2.2: Inheritance
  - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java: Chapter 5 Practice Quiz"

    Use this practice quiz to review.

  - Practice: Understanding Inheritance in JavaQuiz

    Complete this practice assignment, which covers encapsulation and polymorphism. It builds on the last assignment. Note the special use of 'super', in the constructor of the subclass, to call the constructor of the superclass.

Unit 3: C++ Standard Template Library

Nearly every C++ programmer uses the C++ Standard Template Library (STL), a powerful and highly useful library of generic-typed data structures and algorithms. In this unit, we will learn how and why the STL was originally developed. The unit will also introduce you to the history and basics of templates and generic programming before presenting the structures (Containers, Iterators, and Functors) and algorithms that the Standard Template Library contains. By the end of this unit, you will be familiar with the STL, its uses, and its structures and algorithms.

Completing this unit should take you approximately 7 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the history of Generic Programming and the motivation behind the Generic Programming development;
  - demonstrate an understanding of the detailed components used in Generic Programming; and

- demonstrate an understanding of the basic components used in the Standard Template Library with C++.

- 3.1: History and Motivation
  - *Ada Programming*: "Generics"Page

    Read this section for an explanation of the history and purpose of generic programming and the precursor language, Ada. Ada is an Object-Oriented programming language used predominantly in military applications. It shares many of its features – including its structure and typing, with C++. While both languages were developed in the same year, Ada's rapid adoption by certain circles within the Computer Science world led computer scientists to modify C++ by adopting some of Ada's features in subsequent C++ releases. These changes are seen as an important evolutionary step in Object-Oriented programming. These materials cover generic programming, precursor Ada, and ANSI/ISO.

  - John DeNero's "Composing Programs"Page

    A software engineering version of the general computation has four fundamental states: requirements, design, code, validation; or in process terms, defining a problem or task, designing a solution, coding the solution, and testing, the solution. Programming paradigms support the transition from designing to coding. To exploit the similarity among problems and tasks, it is beneficial to reuse existing designs and code on new problems and tasks. The principle concepts of modularity, abstraction and composition support reusability. This article shows how programming languages implement these principles using data, expressions, functions, and control statements.

- 3.2: Main Design Ideas
  - Massachusetts Institute of Technology: Chris Terman's "Basics of Information" and "The Digital Abstraction"Page

    Watch these lectures. These videos are part of another course. For our course, focus on the hardware hierarchy introduced in the first 20 minutes of lecture 1 and the first 34 minutes of lecture 2. These videos reinforce the concepts of modularization, abstraction, composition, and hierarchy, but it does so by stepping 'aside' to take a look at computer hardware.

- 3.3: Elements of C++ STL
  - Alexander Stepanov's "STL and Its Design Principles"Page

    Watch this video, which discusses the principles of the C++ STL (Standard Template Library) and describes abstraction as it pertains to algorithms and data.

There are different kinds of abstraction. For hardware, we used part-whole abstraction to decompose a computer into its module hierarchy. In this lecture, you encounter part-whole abstraction for data, but one based on generality for algorithms – when an algorithm is a special case of another algorithm, the latter is more general than the former. The discourse leads to object-oriented and generic programming.

STL is an organized collection of generic algorithms applicable to a more general collection of problems and tasks. STL templates include Containers, Iterators, Algorithms, and Functors. In this long, but illuminating video, the lecturer gives his insight into the development of STL, its benefits and limitations. Focus on the parts of the lecture that relate to abstraction and composition, and the process of creating programs.

- 3.3.1: Containers and Iterators
  - Jeremy Hansen's "The Rook's Guide to C++"Page

    Read chapter 23, which covers containers and iterators.

  - Wikipedia: "Standard Template Library"Page

    Read this article, which further discusses the STL.
- 3.3.2: Complexity and Cost
  - Massachusetts Institute of Technology: Eric Grimson and John Guttag's "Complexity"Page

    In this unit we continue our exploration of abstraction with respect to algorithms. This lecture discusses the classification of algorithms according to how the performance of an algorithm grows relative to the size of the problem or task the algorithm solves or performs. Algorithms are classified by average run-time complexity, defined as the average number of steps the algorithm takes for a problem of size 'n'. Abstraction ignores the implementation of the algorithm and only considers the growth in the number of (primitive) steps an algorithm takes as the size of the problem grows. The video lecture introduces big 'O' notation and gives examples for linear, log, quadratic, and exponential complexity. The lecturer states that exponential complexity should be avoided in general.
- 3.3.3: Functors
  - Wikipedia: "Function Object"URL

    A functor is a name for a function object, which is an object that is called like a function.

Unit 4: Exceptions

A programmer must ensure the safe execution of his or her code. In other words, if an error occurs, the program should present the user with relevant information and then quit gracefully. The built-in C++ structures used to accomplish this goal are known as "Exceptions". This unit will introduce you to the concept of throwing and catching Exceptions when something goes wrong with your software, explain how they are used in both C++ and Java, and teach you how to handle them. By the end of this unit, you will be able to write safe programs designed to perform gracefully when/if errors occur.

Completing this unit should take you approximately 4 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the roles of exceptions and exception handling in programming;
  - demonstrate an understanding of how exceptions are handled in C++ and Java; and
  - apply and compare exception handling in C++ and Java programs.

- 4.1: The Role of Exceptions

  - Cave of Programming: "Handling Exceptions"Page

    An exception is a runtime error that interrupts the execution of a program statement. When an exception occurs, the runtime system transfers control to either a runtime routine that handles the exception by printing a message and then terminating the program or to a programmer-provided routine that handles the exception by executing instructions that take corrective action. This video gives a Java example of an exception handler using 'throw' and 'catch' blocks.

  - 4.1.1: Traditional Error-Handling Methods

    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

      Read section 8.1.

      An error can be a syntax error, a runtime error, or a logic error. The language compiler or interpreter can detect syntax errors. A runtime error is an unexpected event involving incorrect semantics and interrupts the execution of an instruction. A logic error results in program behavior that does not satisfy the requirements, e.g. an incorrect solution. The resource reading describes some semantic errors that Java can prevent and some that Java can not detect. In the latter case, the Java developer should build program features that either avoid or detect those errors.

  - 4.1.2: Using Exceptions to Write Correct Programs

    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

Read section 8.2 for an explanation of how to prevent errors by writing correct programs and writing program statements to take corrective action or to prevent errors.

Runtime errors and incorrect results when executing a program are addressed by detecting them and taking corrective action when they occur, or by preventing them from occurring in the first place. Corrective action and prevention depend on an understanding of the programming process and the identification of the root cause of errors. The programming process consists of several stages: specifying the requirements for the program, designing the program, implementing the program using a programming language, testing the program, and deploying the program for operation by users.

Errors should be detected as early as possible in the programming process, and each process stage should have a verification activity: requirements, design, code, and even tests should be verified for correctness. Each stage should also be validated (that is, checked that they satisfy the requirements) to prevent errors.

- 4.2: Exceptions in Java and C++
  - 4.2.1: Exceptions in Java
    - W3Resource: "Exceptions in Java"Page

      This article article illustrates the Java flow to handle an exception. Java uses 'try' and 'catch' blocks for the code that handles the exception, and a 'throw' keyword used in the method where the exception occurs. There are 4 sections to the reading, each after the first accessed by clicking <next>. Be sure to read each.

    - TheJavaWorld: "Java Error Handling"Page

      There are several design decisions and are addressed in the design activity of the programming process, including what exceptions to handle, where to place the exceptions handlers, the correspondence of try's to 'catches', the nesting of exception handlers, and which methods will have the throw keywords.

    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

      Read section 8.3, which is a continuation of the previous sections on correctness and robustness, and elaborates on exceptions in Java.

  - 4.2.2: Exceptions in C++
    - *C++ Programming*: "Exception Handling"Page

      Read this overview of the role of exceptions generated by the C++ library. This section also covers throw and try/catch concepts.

Unit 5: Recursion

Often considered one of the more conceptually difficult concepts within the field of Computer Science, recursion – the act of a function invoking itself – is a powerful and relevant tool for any Computer Scientist. In this unit, we will take an in-depth look at recursion, learning the recursive steps, the role that recursion plays in common data structures, and what happens inside the computer when a recursion function is invoked. By the end of this unit, you will recognize situations that require recursion and be able to apply it appropriately.

Note: Recursion can be a difficult concept for some students new to the field of Computer Science. This anxiety is best resolved through the use of an example. For this module, we will employ the use of recursion to write a program to express the "factorial" function. This straightforward example will give the student an overview of all the major components of recursion.

Completing this unit should take you approximately 5 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the general uses of recursion in programming, including its utility in solving programming problems; and
  - demonstrate an understanding of the basic search algorithms that are recursive in nature and how they are used in programming.

- 5.1: Definition

  - Boundless: "Recursive Definitions"Page

    Read this article about recursion.

  - 5.1.1: Divide and Conquer

    - Massachusetts Institute of Technology: John Guttag's "Recursion"Page

      This video discusses recursion as a divide and conquer problem solving technique, a design technique, and a programming technique.

  - 5.1.2: Applying Recursion

    - Massachusetts Institute of Technology: John Guttag's "Recursion Lecture Notes"URL

      These notes expand on the topic of recursion as a way to decompose problems into subproblems. They also apply to decomposing data structures into sub-substructures. Identifying the recursive problem or data structure and their implementation in a program require a thorough understanding of the programming process. Some implementation topics are pointed out, including reentrant code, recursion vs. iteration (loops), relation of recursion to the functional and imperative programming paradigms, and common mistakes in programming recursion.

- Wikipedia: "Tail-Recursive Functions"Page

  Tail-recursion is a type of recursion where the last statement executed is the recursive call and, therefore, there are no instructions after that call. When the return from the recursive call is made, the calling function terminates, in which case the return address to the calling function is not necessary. Some compilers and some languages do not save the return address back to the recursive calling function and translate tail recursion into machine instructions for a loop, which saves space on the stack and execution time. However, implementation of source tail recursion using machine code for a loop is compiler and/or language dependent.

5.1.3: Recursive Structures

- Massachusetts Institute of Technology: Eric Grimsom and John Guttag's "Divide and Conquer Methods"Page

  This video delves deeper into divide and conquer algorithms. Some of these algorithms are recursive. Recursion is a way of decomposing a problem into smaller or simpler problems of the same type. We have discussed the base subproblem and the inductive subproblem when applying recursion. A third consideration is composition – namely, how will simpler subproblems be composed into a solution to the larger problem? Sometimes the decomposition is easy but the composition is hard; sometimes the reverse holds.

  The lecturer employs complexity to compare algorithms – how does the number of steps in a solution to a problem grow as n grows, where n is the number of operations or data elements involved in the inductive step? Formally, that complexity approach is called Big 'O'. The lecturer presents two examples. The message of the video is that designing and implementing algorithms involves tradeoffs: decomposition vs composition tradeoff (merge sort example), and tradeoff of storage space vs. run-time (hash function example).

  Most of the concepts you encounter in programming languages are related to reuse of algorithms or designs and implementations. These are categorized by complexity (for example the merge sort example has n log n complexity), which helps us decide their appropriateness for certain kinds of problems. The lecture closes with enhancements to the examples using exceptions and assertions (pre and post conditions), covered in unit 4 of our course. They help us handle different expectations that may arise in reusing the algorithms.

5.1.4: Recursive Steps

- *Programming via Java*: "Recursion"Page

  This resource goes into recursion for Java in detail. It goes over several Java examples. It states that the steps of induction can be implemented using either recursion or loops; for a particular program, one implementation may be more

effective or efficient than the other. Given a program that implements recursion, stepping through the program statement by statement helps to understand the details of how recursion works at run-time. The run-time system makes use of an internal stack that records the run-time states of a recursive function. Diagrams of the stack are given that show the changes in state of the method during run-time, that is, the pushing of the state when the method is called and the popping of the stack when the method has finished executing. The diagrams help you to understand recursive behavior.

- 5.1.5: Examples of Recursion
  - Khan Academy: "Recursive Factorial Function" and "Fibonacci Numbers"Page

    In these lectures we use Python, which is an easy-to-use procedural and object-oriented language, but our focus will not be on the syntax of the language. Rather, our focus is the concept of recursion, the requirements for the program (i.e., the statement of the problem), the design of the program, the semantics of the program (this is the stage in which we don't worry too much about the syntax of Python – knowing Java and C++ enables you to learn Python easily), and the verification of the program implementation (we do this by stepping through the program and running several test cases).

    Note that the term verification refers to the correctness: the code running without errors and the code correctly implementing the design. Validation refers to the satisfaction of the requirements for the program: the requirements accurately reflecting problem or task the program is to perform, the design satisfying the requirements, and the code satisfying the requirements.

- 5.2: Recursive Algorithms
  - Recursion is a decomposition/composition problem solving technique. With reference to the typical programming process, recursion applies to design and to coding. Recursive algorithms are used in designing, and recursive implementations are used in coding. Usually recursion in design originates from recursive data structures, because they are generic and apply to many types of problems or tasks. For example, trees are a widely useful data structure, used in games, decision making, analysis, etc. Recursive implementations tend to pertain to specific problems or tasks. However, if there are similar problems, a recursive implementation may be applicable to those similar problems. Scan the resource material for Unit 5 looking at the examples used to explain and illustrate recursion, noting how you might use them as problem solving resources for future problems you may want to solve.

Unit 6: Searching and Sorting

As a computer programmer, you will need to know how to search and sort data. This will require you to leverage what you have learned in a number of different Computer

Science areas, drawing from your introduction to data structures and algorithms in particular. In this unit, we will identify the importance of searching and sorting, learn a number of popular searching and sorting algorithms, and determine how to analyze and appropriately apply them. By the end of this unit, you will recognize instances in which you need a searching or sorting algorithm and be able to apply one efficiently.

Completing this unit should take you approximately 9 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate an understanding of the basic components of search algorithms and their various implementations;
  - demonstrate an understanding of the differences between various search algorithms, including list search and tree search;
  - differentiate various sorting algorithms; and
  - demonstrate an understanding of the basic techniques involved in complexity analysis.

6.1: Search Algorithms

- Why study searching and sorting? There are two reasons for doing so.

  First, searching and sorting are tasks that occur frequently and are needed in programming.

  Secondly, the requirements for these two 'needed' functions are simple and clear, their designs (that is, algorithms) are well developed, they are implemented in every language and are available in many code libraries, and their performance (time and space) are well understood.

  Divide and Conquer algorithms, such as List and Tre Search, and Merge and Insertion Sort, make good examples for demonstrating the concepts of this course: decomposition, abstraction, modularization, hierarchy.

6.1.1: List Search

- Massachusetts Institute of Technology: John Guttag's "Memory and Search Methods" and "Binary Search, Bubble, and Selection Sorts"Page

  Sorting usually makes use of search. Watch these lectures on linear and binary search, and note how the use of sorting can also improve search performance.

- Chess Programming Wiki: "Linked List"Page

  Assume we have a collection of data objects, such as telephone numbers, and that we need to find a particular phone number in that collection. We will need a data structure for storing the objects. One such data structure is a list. A list is a generic object and can be used for any type, a type built into our programming language

or a programmer defined object. For example, we can have a list of integers or a list of telephone numbers.

A list is composed of elements and has functions or methods that apply to a list, in particular, insert and remove, which add or delete elements of the list, respectively. Some languages may also have a find function. However, if our language has no such function we will need to write it. This resource discusses the implementation of a program to search a list to find a particular element in the list. Glance at the list of External Links at the bottom of the page; the elements or nodes of the list are grouped by language: Python, Java, C++ , and so on.

- 6.1.2: Tree Search

  - *Composing Programs*: "Recursive Data Structures"URL

    Read this page. In the previous unit of our course we studied recursive algorithms. Recursion is a concept that also applies to data. Here we look at recursive data structures – lists, trees, and sets. A list is a structure that consists of elements linked together. If an element is linked to more than one element, the structure is a tree. If each element is linked to two (sub) elements, it is called a binary tree. Trees can be implemented using lists, as shown in the resource for this unit. Several examples of the wide applicability of lists are presented. A link points to all the remaining links, i.e. the rest of the list or the rest of the tree; thus, a link points to a list or to a tree; this is data recursion.

    The efficiency of the programming process includes both running time and size of data. The reading resource discusses the latter for recursive lists and trees.

    Lastly, why read the last section on sets? Sets are another recursive data structure and the last section 2.7.6, indicates their connection with trees, namely, a set data type can be implemented in several different ways using a list or a tree data type. Thus, the programming process includes implementation decisions, in addition, to design or algorithm decisions. Each of these types of decisions is constrained by the features of the programming language used. The decision choices, such as which data structure to use, will impact efficiency and effectiveness of the program's satisfaction of the program's requirements.

  - Kamal Rawat's "Basic Tree Traversals"Page

    The use of a tree structure involves traversing or stepping through the elements or nodes of the tree. This page shows how to traverse a binary tree, which we can extend to trees having more than two descendants at each node. Many problems can be modeled by a tree. For example, in chess, the first move can be represented by the root or starting node of a tree; the next move by an opponent player, by the descendent nodes of the root. This decomposition can continue for many levels. Thus, a level in the tree hierarchy represents the possible moves available to one player; and the next level, the possible moves of the opponent player. Each level represents the choices available to a given player. Traversing the

tree involves: from a given start node a player looks-ahead at its descendent nodes (the possible moves), from each of these descendant nodes the player looks-ahead at their descendants (possible responding moves of the opponent player), and so on, continuing to look ahead (planning) to cover as many levels as feasible. Based on the look-ahead information (which gets better the further the look-ahead goes), the player chooses a descendant from the given start node.

- Massachusetts Institute of Technology: Dennis Freeman's "Search Algorithms"Page

  Watch this lecture, which illustrates the role of searching in helping to solve many problems. Searching a tree involves traversing the tree and making a decision at each node as we traverse or step through the tree. Most problems involve making decisions. If we can put a value on the outcome of a decision and if we search for a decision that has a 'best' value, then our decision process would be a search process.

  The lecture points out two common techniques for traversing all of the elements of a tree: breadth first and depth first search. A traversal technique involves deciding which descendant (sub) element to look at next. Note that the starting large decision has been decomposed into a series of decisions as to which descendent element to look at on the next level down in the tree hierarchy.

6.2: Sorting Algorithms

- Notice that the structure of this course is a tree. The following subunits in the Unit 6 sub-tree describe sort algorithms.

Whereas searching takes a data structure as input and outputs an element of the data structure, sorting is more complex, in that it takes a data structure as input and returns a data structure of the same type, but with the elements rearranged. There are a few search algorithms: linear for a list, depth or breadth traversal for a tree, and binary search. There are several sorting algorithms; unit 6.2 presents a number of them.

6.2.1: Merge and Insertion Sort

- Massachusetts Institute of Technology: Eric Grimson and John Guttag's "Divide and Conquer Methods, Merge Sort, and Exceptions"Page

  You watched these videos earlier, but take some time to review the merge sort discussion at the beginning of the first lecture. Again, focus on recognizing abstraction, decomposition, and composition. Rewatch the second lecture, focusing on the bubble and selection sorts.

- Khan Academy: "Sorting Algorithms"Page

The following lectures step through the programming life-cycle process for sorting. The programming process comprises requirements, design, implementation of the design in a programming language, and verification/validation. The last lecture adds maintenance, extending the process to a programming life-cycle process. The states of the process overlap and do not have to be performed sequentially.

- 6.2.2: Quick Sort
  - Massachusetts Institute of Technology: Erik Demaine and Charles Leiserson's "Quicksort, Randomized Algorithms"Page

    This lecture explains the details of the working of quick sort, which is on average 3 times faster than merge sort. The lecture has 3 parts: the first 20 minutes approximately, or first third, gives the explanation – watch that part of the lecture. You should revisit the second third, or second and third parts of the lecture when you are in unit 6.2.5, later in our course.

- 6.2.3: Radix Sort
  - Wikipedia: "Radix Sort"Page

    The radix sort does not compare values of the elements to be sorted; it uses the digits (in some radix, e.g. base 2) of integer keys and sorts the keys by sorting, first on the first digit (either the least significant or most significant digit), then on the next significant digit, and so on, up to the last digit. This decomposes the problem into n smaller sorting problems, namely, sorting, all the values that have the same digit in the same radix position of the key. Read this article on the Radix sort. Carefully study the discussion on efficiency and note that the complexity depends on the assumptions made regarding the primitive steps and the data structures used in a program.

- 6.2.4: Analysis
  - Massachusetts Institute of Technology: Eric Grimson and John Guttag's "Complexity; Log, Linear, Quadratic, Exponential Algorithms"Page

    Understanding the complexity of an algorithm helps us decide whether or not we should use it as the design of a program to solve a problem. Complexity is usually measured in terms of the average number of steps in the computation of a program. The steps can be used to estimate an average bound, lower bound, and upper bound of the amount of time and for the amount of storage space needed for the computation. The lecture explains Big O notation and concept and, using recurrence relations, develops the Big O value for several types of computations. The steps of interest are the primitive steps of an algorithm and the operations that are intrinsic to the data structure used in the program implementation of the algorithm.

Big O estimates are usually associated with the algorithm – its primitive operations and data structure. Algorithms are grouped into classes associated with linear, n log n, quadratic, and exponential Big O bounds: O(n), O(n times log n), O(n**2), and O(2**n), respectively. For a given value of n, the lecturer gives the values of n, n x log n, n**2, and 2**n, to show the dramatic growth as n grows. If a problems grows by a factor of n an algorithm that grows too fast, e.g. has quadratic or exponential growth, would not be a good choice for the design of a program to solve it.

Note that the details of complexity analysis require a strong mathematical background, and you should focus on the main ideas or 'big picture' first before delving into the details.

- "Big O Notation"Page

  The video gives a concise definition of Big O, which is popular for bounding the running time for search and sort algorithms. Additionally, if you feel comfortable with your math background, you should watch the second and third parts of the video.

Unit 7: Template Programming

In Unit 3, we discussed the C++ Standard Template Library and introduced the concept of template programming. In this unit, we will further explore both templates and generic programming, cultivating a broader understanding of the topic. We will begin with a review of the STL and a discussion of the motivation behind template development. We will then discuss templates in a more generic fashion, referring back to the STL where necessary, and take a look at how templates are utilized within Java, comparing and contrasting with the C++ STL. By the end of this unit, you will recognize the importance of templates, be able to identify when they are needed, and know how to apply them efficiently.

Completing this unit should take you approximately 5 hours.

- Upon successful completion of this unit, you will be able to:
  - demonstrate a detailed understanding of generic programming and principles in the standard template library;
  - demonstrate an understanding of various types used within the standard template library in C++; and
  - demonstrate an understanding of various types used within the standard template library in Java.

- 7.1: Generic Programming
  - Massachusetts Institute of Technology: Eunsuk Kang's "Introduction to C Memory Management and C++ Object-Oriented Programming"Folder

Study the slides for lectures 5 and 6 for a good foundation discussion of generic programming concepts.

Polya, a famous mathematician, wrote "How to Solve It", which introduced generic techniques, applicable to solving any problem. Since the objective of the programming process is a solution to a problem or task, perhaps the concepts in Polya's book are applicable. The programming process as we have seen includes requirements development (understanding the problem), design (formulating a plan or strategy), implementation in a programming language (completing the plan by providing details from previous or known solutions), verification and validation (checking the solution). Given a specific problem (program), we can develop a specific solution (specific program); or, we could use abstraction to view the program generically, and develop a generic solution. Using generic functions and generic data structures, a generic design and generic implementation can be developed that will solve, not just a specific problem, but a general class of problems. That is the idea of generic programming; a form of reuse on a large scale, of requirements, design, and implementation.

Templates and Template Libraries are an approach to generic programming. One approach to generic programming is a capability that allows functions that apply to more than one type of data (polymorphism). For example, a generic search should work for searching data of any type, as long as the data is comparable.

- 7.2: Templates in C++
    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

Read this section for an introduction to the generic programming in several languages, including Java and C++. It covers the standard template library, containers, algorithms, templates, and mentions compile time polymorphism vs. runtime polymorphism.

    - Massachusetts Institute of Technology: Jesse Dunietz, Geza Kovacs, and John Marrero's "Introduction to C++"URL

Explore the lecture notes and assignments for this short course, which discuss C++ templates, the Standard Template Library (STL), and operator overloading.

- 7.3: Commonly-Used Classes
    - Hobart and William Smith Colleges: David Eck's "Introduction to Programming Using Java"Page

Read section 10.2. In an class hierarchy, a base class is more generic than a class that implements it. In particular, functions or methods of a base class are more generic functions. While generality is desirable, it may be a trade-off with efficiency for certain problems. This section looks at several classes in the class hierarchy for lists

and for sets, and the additional features implemented in the subclasses that make them more efficient depending on the problem to be solved.

- Practice: Understanding and Creating TreeSets in JavaQuiz

  In this activity, you will construct a basic Java application that contains a TreeSet object. You will also add items to this Treeset.

- Practice: Understanding and Creating HashSets in JavaQuiz

  This activity provides an exercise to enable you to create a basic java application containing a HashSet object.

CS107: C++ Programming

In this course, we will learn the mechanics of editing and compiling programs in C++. We will begin with a discussion of the essential elements of C++ programming: variables, loops, expressions, functions, and string class. Then, we will cover the basics of object-oriented programming: classes, inheritance, templates, exceptions, and file manipulation. We will then review function and class templates and the classes that perform output and input of characters to/from files. This course will also cover namespaces, exception handling, and preprocessor directives. In the last part of the course, we will learn some slightly more sophisticated programming techniques that deal with data structures such as linked lists and binary trees.

Unit 1: Introduction and Setup

This unit presents a brief history of C++ before addressing the mechanics of editing and compiling simple programs in C++ using the Eclipse IDE (integrated development environment). We will focus on how to write and format a general C++ program, the meaning of the `main()` function, how to use the `cout` and `cin` objects, how to declare and use variables, and how to use arithmetic operators.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - describe the basic history of C++;
  - set up and create a simple C++ project using Eclipse CDT;
  - explain the meaning of simple C++ commands;
  - use cout and cin objects effectively;
  - declare and use variables; and
  - use C++ operators.

- 1.1: A Basic History of C++

- The History of ProgrammingPage

  Watch this video for a broad overview of the history of computer programming in general.

- Introduction to C++Page

  Read this article on the history of C++ programming to see how C++ evolved into what it is today.

- 1.2: How to Compile and Run a C++ Program
  - How to Compile a C/C++ Program on Ubuntu LinuxURL

    If you are running a Linux operating system, read this article on how to compile and run a C++ program. Even though you do not yet understand the C++ language, you can follow the operational steps to compile a program from C++ to machine code.

  - Installing the Eclipse CDT, Part 1Page

    Watch this video to learn how to install the Eclipse CDT.

  - Installing the Eclipse CDT, Part 2Page

    For the programming in this course, we will use the Eclipse CDT. In order to follow these steps, you need to first install Eclipse. You can go to https://www.eclipse.org/downloads/ to download the Eclipse software, which then prepares you for write Java code. However, we want to write code in C++ instead of Java, so we have to take a few additional steps that are different depending on your operating system. Those steps are described in this article.

  - "Hello World"Page

    After you watch this video, try to create, compile, and run your own Hello World program.

- 1.3: Simple C++ Commands
  - Understanding the "Hello World" ProgramURL

    After you read this article, compile the example and make sure you understand the code in each line.

- 1.4: Variables, Data Types, and Constants
  - C Variables and Data TypesURL

Read this article about data types. Although this article discusses C, the data types in C++ are exactly the same. The only difference is that C does not include a boolean variable. For now, just note that a boolean variable holds a value of either True or False.

- C++ Variables and Data TypesPage

Watch this video, which explains how data types are used in C++.

- 1.5: Basic Input and Output
  - Taking User InputPage

Watch this (admittedly robotic) video, which explains input and output.

- 1.6: Arithmetic Operators
  - C OperatorsURL

We touched on operators in the previous unit, but now we will cover them in more detail. Even though this is C and not C++, there are no major differences in their operators.

  - Arithmetic Operators in C and C++Page

Watch this video, which discusses the use of arithmetic operators and their real world applications.

- Unit 1 Exercises
  - Basic C++ ExercisesURL

Exercises 1 to 36 cover basic C++ concepts that you should be able to complete if you have worked through all of the materials in unit 1. Start with some early problems, then try a few from the middle, and then a few more later from the list. Do not attempt any of the problems after Exercise 36, as we haven't learned these concepts yet. Don't worry, we will come back to them.

  - Input-Output ExercisesURL

Complete these exercises to check your understanding of input and output.

Unit 2: Structuring Program Code

This unit focuses on implementing simple control structures. First, we will learn how to use conditional and iteration structures to make decisions and to repeat code. We will then discuss how to use debugging tools to test and troubleshoot these structures. We will also explore how to break our code into smaller, more manageable pieces by putting certain common pieces into functions. We will also discus scope, as well as passing variables by value and by reference. Finally, we explore a special type of class, the string,

which has some special functions that allow us to manipulate text. By introducing classes and how they are used here, we will be ready to tackle object-oriented programming in the next unit.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - use conditional and iteration structures in C++;
  - explain the purpose of different features of the debugging tool;
  - define test cases and coverage analysis;
  - use simple functions; and
  - use character arrays and the functions of the string class.

- 2.1: Conditional and Iteration Structures
  -  Flow of ControlFile

    Read these lecture notes to learn about control structures in C++ programming. Compile the examples from the notes, and make sure you understand the code in each line. After reading these notes, you should be able to define conditional structures, iteration structures, and jump statements.

  - If-Else StatementsPage

    Computers are thought to be very intelligent machines, but they are only as intelligent as the humans that program them. If-else gives programs the capability to make decisions and provide an illusion of intelligence. Watch this video to learn how to add this intelligence capability to your programs.

  - Complex ConditionsPage

    Simple decisions are useful on many occasions, but sometimes decisions are more complex and have multiple layers. Here you can learn how to combine if statements with AND and OR to address many different scenarios.

  - While LoopsPage

    While decisions give computers their "intellectual" abilities, loops give them their usefulness. The ability to repeat something over and over for dozens, hundreds, or even millions of times ensures that computer programs can perform tasks that are quite complex in a fraction of the time it takes us humans. The while loop is the fundamental approach to repetitive sequences.

  - Do-While LoopsPage

    The do-while loop is a version of the while loop. Though it is possible that a program will never enter the loop with a while loop (since it is possible that the condition is already met before entering the loop), do-while loops guarantee that

the loop gets performed at least once. Watch this video to learn how do-while loops differ from the standard while loop.

- For LoopsPage

  Although while loops are the basic loop structure, for loops are the enhanced version of while loops that allow for more flexibility, and let programmers simplify and improve the accuracy of their code. Watch this video to learn how to use for loops.

- 2.2: Testing and Debugging
  - Software Unit Test Policy and Coverage AnalysisURL

    Read this section and the one on coverage analysis to learn more about the basics of software testing.

  - Debugging in EclipsePage

    Learning to debug is an essential skill for any programming language. Knowing how to search your code for bugs, walk through the program's interpretation, and examine the variable manipulation is an essential part of debugging. Learn how to use the tools built into the Eclipse environment to debug your program. Though this video refers to Java, debugging C++ in Eclipse works similarly.

- 2.3: The Scope of Variables in a Simple Function
  - Functions in C++URL

    Functions are small segments of code that are removed from the normal code flow, and they are called to perform specific actions. After the function executes, control returns back to the normal flow. Within a function, data may be sent and returned to the rest of the program.

- 2.4: Arguments Passed by Value and by Reference in a Simple Function
  - Passing Arguments by Value and by ReferenceURL

    When data is passed by value, the actual contents of the variable is passed. However, with some data, you may instead simply want the program to access the data directly from the current location where it is stored. To do this, you pass the value by reference, which tells the function to access the address of the variable.

- 2.5: Functions of the String Class
  - StringsURL

Now we'll explore the use of string arrays, which are part of C programming. C did not allow for the use of strings initially, so string arrays are simply a string of characters that are stored in an array. We haven't discussed arrays yet, but this is a great introduction to the use of character strings.

- String ArraysURL

Review this page for some more information on character strings, or string arrays.

- The String ClassURL

The string class is an extension that was added to C++ that deals with character strings as strings, not as arrays. Read this article to learn how to use classes to call methods and how to manipulate strings to access text data.

- Strings and CharactersURL

This chapter highlights the many different functions that are part of the string class in C++.

- Unit 2 Exercises
  - Practice with IterationsURL

    Complete these exercises to test your understanding of iterations.

  - Practice with FunctionsURL

    Complete these exercises to test your understanding of functions.

  - C++ Problem SetURL

    Answer the six questions in problem set 1. After you finish, check your answers.

  -  C String ExercisesURL

    Complete these exercises to test your understanding of string arrays.

Unit 3: Working with Simple Data Structures

Data structures are just ways to store multiple data values. Arrays, structs, enumerations, unions, queues, lists, and vectors are a few examples of data structures. In this unit, we will focus on a few simple structures. The array is one of the most basic structures used in computer programming. Arrays store data contiguously by representing data with a common name and distinguishing it by its index. This is like a parking garage: the garage stores vehicles; all of the parking spaces have a common name (the name of the garage); and each parked vehicle is identified by a specific parking space number. This is also true of arrays. Think of arrays as parking garages for our data. After exploring arrays, we then

learn structs, unions, and enumerations, which are special ways to group more complex data types.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - use arrays; and
  - use struct, unions, and enumerations.

- 3.1: Arrays
  - ArraysURL

    Arrays are the most basic data structure in C++. Read this page and watch the videos to learn about arrays.

  - Multidimensional ArraysPage

    Watch this video to learn about multidimensional arrays.

- 3.2: Structs, Unions, and Enumerations
  - EnumerationURL

    Enumerated types (enums) are essentially a series of int variables that represent more complex data types.

  - StructsURL

    Structs are a great way to pair multiple variables together and allow them to be passed as a group.

  - UnionsURL

    Unions are very similar to structs, but the data shares the same memory spaces, so they cannot be used at the same time.

- Unit 3 Exercises
  - Practice with ArraysURL

    Complete these exercises to test your understanding of arrays.

  - More Practice with ArraysURL

    Complete the assessments at the bottom of this page to test your understanding of arrays and how to use them.

Unit 4: Object-Oriented Programming

In this unit, you will learn how to design a class, which is an expanded concept of a data structure that can hold both data and functions. An object is an instantiation of a class, so a class would be the type, and an object would be the variable. Next, we will learn how to handle private and protected members of a class, which is important for sound class design. This unit covers a key feature of C++ classes: inheritance. Inheritance allows classes to inherit objects and functions from other classes. In this unit, we will learn how classes can inherit members from more than one class. We will end this unit with the study of polymorphism or the ability to create a variable, a function, or an object that has more than one form. This brings object-oriented methodologies to their full potential.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - define and use constructors and deconstructors;
  - create overloading operators;
  - define and use the keyword "this" and use the static members appropriately;
  - design and appropriately use friend functions and classes;
  - use class inheritance to design better code;
  - explain how polymorphism is achieved through C++ code; and
  - create accessors, mutators, and access modifiers.

- 4.1: Class Design
  -  Object-Oriented Programming (OOP) and InheritanceFile

    Classes are what separate the capabilities of C from that of C++. Classes are the object-oriented aspect of programming.

    To think about classes, consider writing utensils. A pen, a pencil, and a marker are all writing utensils. However, when a pen uses ink; a pencil uses graphite, and so forth. The color of the pencil or the pen might be different, but their shapes and behaviors are the same. This is what object-oriented programming is about. We can define the characteristics of a writing utensil and then create specific instances of each object based on the unique characteristics of each instance of the object, without having to define their common characteristics each time.

  - Classes and ObjectsPage

    This video is a great demonstration of the use of classes.

- 4.2: Inheritance between Classes
  - InheritancePage

    Watch this video on inheritance between classes.

  - Basics of Inheritance in C++ With ExamplesURL

This article demonstrates some examples of inheritance.

- Static AttributesURL

  Static attributes allow us to store one value for a variable that is consistent through all instances of a class. Read this article to see how this is implemented.

- 4.3: Polymorphism
  - Encapsulation, Inheritance, and Polymorphism In C++URL

    This article reviews inheritance and explains polymorphism.

  - PolymorphismPage

    Polymorphism is not easy to grasp. This video provides a solid explanation of the concept.

  - Operator OverloadingURL

    Operators play an important role in computer programming. By using it, you can change the behavior of an operator based on the types of its arguments.

  - Encapsulation and PolymorphismFile

    Review these slides, which summarize the concepts you have been learning thus far.

- Unit 4 Exercises
  - Practice with InheritanceURL

    Complete this assignment by copying the text into Eclipse. Compile and run the code, and then complete assignment 6. Create, compile, and run the code to check your understanding of this concept. The instructions for both provide details for running in a command line environment. Ignore this information and create the file in your Eclipse environment as you usually do.

Unit 5: Advanced Concepts

In this unit, we will first explore the concept of generic programming, and how you can harness the power of templates to make classes and functions more reusable and adaptable to your personal needs. Then, we will explore ways to add content to our program by reading from files and then storing the data by writing to output files. Finally, we will explore the use of exceptions in C++. Exceptions allow us to anticipate problems that might occur in code and handle this problems through the use of exceptions, telling the program specifically how it is to behave when these problems occur.

Completing this unit should take you approximately 8 hours.

- Upon successful completion of this unit, you will be able to:
  - write class and function templates;
  - code with a class that manipulates files; and
  - use exceptions in C++ code.

- 5.1: Writing Class and Function Templates

  - C++ TemplatesURL

    Read this article, which discusses generic programming and the use of templates.

  - Introduction to C++ TemplatesPage

    Watch this video to learn more about templates.

- 5.2: Inputting and Outputting with Files

  - Reading File Input in C++Page

    The ability to read from a file is a critical task that allows for batch processing. Watch this video to learn how you can read data from a file, so that data input doesn't always have to come from the keyboard.

  - Output File Streams in C++Page

    Sometimes once you have finished processing your data, you want to save the data so that it can be re-input at a later time as an input file. Watch this video to learn how to write to an output file using output file streams.

  - Input and OutputURL

    Read this article and complete the exercises at the end to test your skills in inputting from and outputting to files.

- 5.3: Exception Handling

  - Exception HandlingURL

    When things go wrong in programming, as they usually do, the last thing you want is for the program to crash or perform some erroneous actions. Using exception handling allows the programmer to control what happens when the unexpected happens.

- Unit 5 Exercises

  - Practice with File HandlingURL

    Complete these activities to practice reading from and writing to files.